Express 中间件原理

在 Node 开发中免不了要使用框架，比如 express、koa、koa2

拿使用的最多的 express 来举例子

开发中肯定会用到很多类似于下面的这种代码

```
var express = require('express');
var app = express();
app.listen(3000, function () {
    console.log('listen 3000...');
});

app.use(middlewareA);
app.use(middlewareB);
app.use(middlewareC);
```

对我要说的就是 app.use()

app.use()就是通常所说的使用中间件

那中间件是什么呢？它又有啥用呢？

中间件 middleware

一个请求发送到服务器后，它的生命周期是 先收到 request（请求），然后服务端处理，处理完了以后发送 response（响应）回去

而这个服务端处理的过程就有文章可做了，想象一下当业务逻辑复杂的时候，为了明确和便于维护，需要把处理的事情分一下，分配成几个部分来做，而每个部分就是一个中间件。

app.use 加载用于处理 http 请求的 middleware（中间件），当一个请求来的时候，会依次被这些 middlewares 处理。

中间件执行的顺序是你定义的顺序

那中间件到底是个什么东西呢？

中间件其是一个函数，在响应发送之前对请求进行一些操作

```
function middleware(req,res,next){
    // 做该干的事

    // 做完后调用下一个函数
    next();
}
```

这个函数有些不太一样，它还有一个 next 参数，而这个 next 也是一个函数，它表示函数数组中的下一个函数

那函数数组又是什么呢

express 内部维护一个函数数组，这个函数数组表示在发出响应之前要执行的所有函数，也就是中间件数组

使用 app.use(fn)后，传进来的 fn 就会被扔到这个数组里，执行完毕后调用 next()方法执行函数数组里的下一个函数，如果没有调用 next()的话，就不会调用下一个函数了，也就是说调用就会被终止

Express 中间件的使用

理论部分简单的说了一下，现在来用代码验证一下，注意需要安装一下 express

```
var express = require('express');

var app = express();
app.listen(3000, function () {
    console.log('listen 3000...');
});

function middlewareA(req, res, next) {
    console.log('middlewareA before next()');
    next();
    console.log('middlewareA after next()');
}

function middlewareB(req, res, next) {
    console.log('middlewareB before next()');
    next();
    console.log('middlewareB after next()');
}
```

```
function middlewareC(req, res, next) {
    console.log('middlewareC before next()');
    next();
    console.log('middlewareC after next()');
}

app.use(middlewareA);
app.use(middlewareB);
app.use(middlewareC);
```

输出结果：

```
→  node-express-middleware-study git:(master) ✗ node server1.js
listen 3000...
middlewareA before next()
middlewareB before next()
middlewareC before next()
middlewareC after next()
middlewareB after next()
middlewareA after next()
```

可以看到在执行完下一个函数后又会回到之前的函数执行 next()之后的部分
这可以理解为中间件的一个特性吧

现在可以说已经明白 Express 的中间件是什么了，以及 app.use 的用法了，下面就来自己实现一下吧

实现简单的 Express 中间件

```
var http = require('http');

/**
 * 仿 express 实现中间件机制
 *
 * @return {app}
 */
function express() {

    var funcs = []; // 待执行的函数数组

    var app = function (req, res) {
        var i = 0;
```

```
        function next() {
            var task = funcs[i++];  // 取出函数数组里的下一个函数
            if (!task) {    // 如果函数不存在,return
                return;
            }
            task(req, res, next);   // 否则,执行下一个函数
        }

        next();
    }

    /**
     * use 方法就是把函数添加到函数数组中
     * @param task
     */
    app.use = function (task) {
        funcs.push(task);
    }

    return app;    // 返回实例
}

// 下面是测试 case

var app = express();
http.createServer(app).listen('3000', function () {
    console.log('listening 3000....');
});

function middlewareA(req, res, next) {
    console.log('middlewareA before next()');
    next();
    console.log('middlewareA after next()');
}

function middlewareB(req, res, next) {
    console.log('middlewareB before next()');
    next();
    console.log('middlewareB after next()');
}

function middlewareC(req, res, next) {
    console.log('middlewareC before next()');
```

```
    next();
    console.log('middlewareC after next()');
}

app.use(middlewareA);
app.use(middlewareB);
app.use(middlewareC);
```

JS 是一门神奇的语言，这里用到了两个闭包，并且给 app 这个函数添加了一个 use 方法，函数也是可以有属性的

原理就是每调用一次 use，就把传进来的函数扔到 express 内部维护的一个函数数组中去

测试结果：

ok，相信对 Express 中间件的原理已经有所了解了。